SBOL-OWL: An ontological approach for formal and semantic representation of synthetic biology information

Göksel Mısırlı,[*,†] Renee Taylor,[†] Angel Goñi-Moreno,[‡] James Alastair McLaughlin,[‡] Chris Myers,[¶] John H Gennari,[§] Phillip Lord,[‡] and Anil Wipat[‡]

[†]*School of Computing and Mathematics, Keele University*

[‡]*School of Computing, Newcastle University*

[¶]*Department of Electrical and Computer Engineering, University of Utah*

[§]*Department of Biomedical Informatics and Medical Education, University of Washington*

E-mail: g.misirli@keele.ac.uk

Abstract

Standard representation of data is key for the reproducibility of designs in synthetic biology. The Synthetic Biology Open Language (SBOL) has already emerged as a data standard to represent information about genetic circuits, and it is based on capturing data using graphs. The language provides the syntax using a free text document that is accessible to humans only. This paper describes SBOL-OWL, an ontology for a machine understandable definition of SBOL. This ontology acts as a semantic layer for genetic circuit designs. As a result, computational tools can understand the meaning of design entities in addition to parsing structured SBOL data. SBOL-OWL not only describes how genetic circuits can be constructed computationally, it also facilitates the use of several existing Semantic Web tools for synthetic biology. This paper demonstrates

some of these features, for example, to validate designs and check for inconsistencies. Through the use of SBOL-OWL, queries can be simplified and become more intuitive. Moreover, existing reasoners can be used to infer information about genetic circuit designs that cannot be directly retrieved using existing querying mechanisms. This ontological representation of the SBOL standard provides a new perspective to the verification, representation, and querying of information about genetic circuits and is important to incorporate complex design information via the integration of biological ontologies.

Keywords

ontology, Synthetic Biology Open Language, standardization, genetic circuits

Synthetic biology deals with the rational design, and implementation, of novel biological functions in living systems.[1] Applications of such engineered systems often rely on genetic circuits that include different biological parts and complex relationships between them.[2] Reproducibility of these circuits is often challenging due to not having complete information about the designs.[3] Moreover, design information is often captured using free text, which aids understanding for humans but may be open to interpretation. It is crucial that designs are unambiguously represented and sufficient information is provided for the sake of reproducibility.[4,5] This process requires not only capturing data using a common syntax but also agreeing on semantics for machine interoperability. The use of computationally tractable representations of designs, for instance via the GenBank format,[6] often omit essential information about the descriptions of gene products, such as molecular interactions and hierarchical structure. Visual representations are useful for communicating design function, but are oversimplified and do not reveal detailed information. As the size and complexity of designs increase, it becomes even more important to use automated approaches and to provide machine accessible data.

The Synthetic Biology Open Language (SBOL)[7,8] has emerged as a data format to elec-

tronically exchange information about genetic circuits. By using SBOL, data about biological roles, structures and molecular interactions of biological parts such as proteins, DNA, RNA, small molecules, and complex molecules can be captured. A single protein, a signalling molecule, or a genetic element such as a promoter or coding sequence (CDS), whether it exists naturally or it is constructed synthetically can be represented as an SBOL entity. SBOL facilitates the reuse of information via modularity. For example, genetic circuits, built from individual genetic elements, can be represented hierarchically using part-subpart relationships. That is, a parent part can include several child components, which in turn can be composed of other parts. This approach is ideal for engineered or natural genes, composed of several genetic elements. The same approach can also be extended for multiple genes, whole plasmids and genomes. Information about these circuits can then be linked to computational models, experimental data and lab protocols using SBOL. In addition, combinatorial libraries and oligo pools can represented in SBOL. Abstract genetic circuit designs can be partially defined without specifying exact sequences. A similar part-subpart relationship is also used to capture molecular interactions in the form of hierarchical modules. For example, Roehner and colleagues[9] demonstrated this idea to encode a CRISPR-based regulatory module with inputs and outputs in order to facilitate its reuse in other designs. This representation is a graph structure with repeated relationships between parent and child components. Graphs are formed of nodes and edges. Nodes can represent entities in a specific domain and edges represent the relationships between these nodes. This approach is ideal to capture complex design information. As a result, the SBOL community adopted a graph-based representation of data, in the form of Resource Description Framework (RDF) documents (`https://www.w3.org/RDF`).

RDF documents are simply graphs, in which domain specific entities can be globally identified using Uniform Resource Identifiers (URIs, `https://www.ietf.org/rfc/rfc3986.txt`) and can have properties whose values are typically literals, such as strings and integers. Values can also be URIs pointing to other entities. As a result, RDF graphs are defined

with triples that include information about an entity of interest, a value, and how this entity and the value are associated. For example, the 'ptetR precedes B0034' triple indicates that the 'ptetR' entity (*subject*) is linked to the 'B0034' entity (*value*) with the 'precedes' relationship (*property*). By incorporating URIs, this triple can be represented as 'pr:ptetR sbol:precedes pr:B0034', where 'pr': and 'sbol:' stand for '`http://parts.igem.org`' and '`http://sbols.org/v2#`' respectively. The use of these URIs indicate that the 'precedes' term from SBOL is used to describe how two entities from the Registry of Standard Biological Parts[10] are related to each other in a genetic circuit. These URIs may not be accessible and may simply act as unique identifiers in an RDF graph.

Using the RDF approach allows SBOL to be flexible when capturing different types of information about genetic circuits, in the form of nodes and edges. This flexibility mainly comes from being able to store a graph in multiple ways and the use of unique URIs. For example, a child RDF entity, identified with a unique URI, can either be embedded within a parent entity, or be referred to using this URI. In fact, the child entity and the parent entity can be stored in different repositories. The use of these URIs also facilitates data aggregation, since two nodes with the same URI are regarded as a single node. When the 'pr:ptetR sbol:precedes pr:B0034' triple is used together with the 'pr:B0034 sbol:precedes pr:luxR' triple, the resulting graph contains three nodes and two edges, since the *value* node of the former triple is the *subject* node of the latter. Therefore, when different types of design information about a genetic circuit is uploaded into graph repositories, the information can be integrated implicitly at the graph level. However, RDF only defines a data model to represent information using graphs. Tools are still left with the big task of interpreting this information. As the amount of data produced by different research groups in different geographical locations increases, it is becoming more important to capture properties linking different entities and values semantically to make the design information machine understandable. Adopting a graph representation of data also brings the advantage of being able to incorporate custom metadata or annotations. These annotations can either be embedded

within SBOL entities to provide additional information or can be used to describe custom domain specific entities. However, the semantics of such metadata are not captured by the SBOL specification. It is therefore important to semantically identify non-SBOL entities and to facilitate their formal representation where possible.

Once a genetic circuit design is constructed, it is crucial to verify the design by querying various design constraints, which may affect the functionality of a circuit.[11] For example, a CDS should be preceded by a ribosome binding site (RBS) for genetic production. Similarly, if a promoter requires transcriptional activation then the intended interaction and the transcription factor (TF) should be represented in the design. Graph representation is ideal for SPARQL[12] (`http://www.w3.org/TR/rdf-sparql-query`) queries (see Figure 7A), which are themselves graph patterns that report matching graph data. However, these queries are created using the RDF syntax and specify exact relationships between nodes and edges, if corresponding data models are not captured semantically. On the other hand, logical queries can facilitate writing simpler and intuitive queries that can be used instead of graph-based queries to integrate querying information over multiple types of nodes and edges. Moreover, graph-based queries may not be ideal where hierarchical data are represented through complex relationships, which is often the case for SBOL.

There have been debates about the particular format of the serialization for SBOL documents (see `http://sbolstandard.org/development/meetings` for some details). In his 2010 dissertation, Galdzicki considered developing a full OWL representation of SBOL, including strong semantics.[13] However, at the time, this approach seemed heavy-handed since the SBOL data model was small and included entities to represent DNA level information only. As a result, this OWL-based approach was not pursued once the community agreed on a graph serialization. As a graph language, RDF has different formats, one of which is eXtensible Markup Language (XML). The RDF/XML format has the advantage of representing graphs for XML tools. As a result, the SBOL community agreed on the RDF/XML serialization to utilize both RDF and XML tools. The rules defining this serialization process

is defined in SBOL specifications. These documents are produced as a community effort led by the SBOL chair and editors, and the SBOL Steering Committee. As of this writing, the specification is 132 pages (Version 2.2.1), and grows incrementally between different versions. Currently, SBOL data are created using bespoke software libraries and there is no formal representation of the rules to construct SBOL compliant descriptions of genetic circuit designs. Although the RDF/XML format facilitates the graph-based representation of genetic circuit designs, whilst supporting XML tools, providing formal definitions of SBOL entities and their relationships remains an issue.

Currently, rules to validate SBOL documents are written in a free-text specification. These rules are interpreted by software developers, and programmatic validation strategies are applied. Libraries have already been developed for reading and writing SBOL entities in Java,[14] C, Python,[15] and JavaScript[16] programming languages. However, the use of such bespoke software libraries is not always necessary to work with SBOL information. As RDF graphs, genetic design information captured using the SBOL data model is already exposed to some of the existing Semantic Web[17] resources. Triplestores are used to store design information, and provenance of designs are captured using previously developed Semantic Web resources. The SynBioHub[18,19] design repository, for example, is based on an RDF triplestore, and, hence, standard SPARQL[20] graph queries are used to extract design information.

It is desirable to semantically enrich the SBOL specification for machine access, which can facilitate the interpretation of SBOL terms computationally in order to utilize different tools. Moreover, this process should be flexible to facilitate the development of extensions and to formally capture additional biological design information. Ontologies can provide computable semantics, which can be used by tools to infer new information about design components and their relationships. Gruber defines an ontology as "*an explicit and formal specification of a conceptualisation*".[21] An ontological representation of the SBOL data model can also be used to provide a shared understanding of design information in order to capture

different entities consistently and unambiguously for machine access.

Ontologies have already been widely used to model biological knowledge and to infer information.[22–25] However, SBOL entities and their relationships, including those with external terms, are not formally defined in other ontologies. SBOL currently utilizes external ontological terms to standardize the meaning of some of the design information. The SBOL specification[8,26] recommends the use of a set of terms from different ontologies, and lists these terms in human readable tables. These terms often act as values for SBOL specific entities. For example, the `type` property of a DNA component in SBOL should come from the Biological Pathway Exchange (BioPAX) ontology.[27] It is desirable to provide formal definitions of SBOL entities and their relationships ontologically in order to bridge the use of different biological ontologies and genetic design information.

This paper presents SBOL-OWL, an ontological representation of the SBOL specification for machine access. Our goal is to bring a new perspective to the verification, representation, and querying of information about genetic circuits using this ontological approach. The development of libraries has been a key in the adoption of the standard. SBOL-OWL brings another opportunity of using readily available ontological tools such as automated reasoners and ontology editors.[28–30] This semantic representation of SBOL data also allows richer queries to be expressed in a simpler and more logical manner. Domain specific information can hence be represented as logical axioms that are formed of multiple graph nodes and edges. With this approach, users can query the data more intuitively, rather than using complex graph structures to extract information. Moreover, SBOL-OWL facilitates writing ontological queries that cannot be supported using a graph-based approach. Examples of such queries, particularly to create recursive queries to fetch information from hierarchical designs, are shown in the results section. SBOL-OWL also facilitates formalizing the SBOL specification for machine interoperability. Such machine access can be used to represent designs semantically, find inconsistencies, execute richer queries, visualize design information using semantic graphs and track changes between different versions of the SBOL standard.

Results and Discussion

The SBOL-OWL ontology has been developed to provide machine accessible description of the SBOL data standard. The ontology is encoded using the Web Ontology Language (OWL, `http://www.w3.org/2004/OWL`),[31] and comes with standard terms or classes. Classes are basic units of ontologies[22] and are used to define types of objects in a domain. Here, they represent SBOL specific entities. SBOL-OWL does not change how SBOL is currently used. Instead, it provides a semantically-aware layer for genetic circuit designs. As a result, Semantic Web tools that process OWL files can now be used with genetic circuit designs. For example, OWL reasoners can infer implicit information about these designs using logical axioms. Similarly, OWL tools can better visualize implicit relationships between SBOL-OWL entities.

In the Semantic Web stack, ontologies are used to provide semantics of domain entities whilst RDF is ideal to syntactically represent information. Here, SBOL specific terms form a basis to formalize the standard in the form of the SBOL-OWL ontology and RDF is used to exchange genetic circuit designs. This semantic layer allows tools to understand complex relationships between individual design components and to carry out subsequent processing, such as querying and integrating data.

Encoding SBOL Entities. SBOL-OWL terms were created using the free text information in the SBOL data standard. Each term has a unique identifier which corresponds to how an SBOL entity is serialized. Free text meanings of these SBOL entities in the SBOL specification are used to populate terms' descriptions. The standard `rdfs:label` and `rdfs:comment` properties are used to represent names and descriptions of these terms respectively.

The most important SBOL entities are categorized as `TopLevel` entities, which include `ComponentDefinition`, `ModuleDefinition`, `Collection`, `Sequence`, `Model`, `Attachment`, `Implementation` and `CombinatorialDerivation` (see Figure 1). As shown in Figure 2, these `TopLevel` entities are regarded as core terms in the ontology, since these entities are

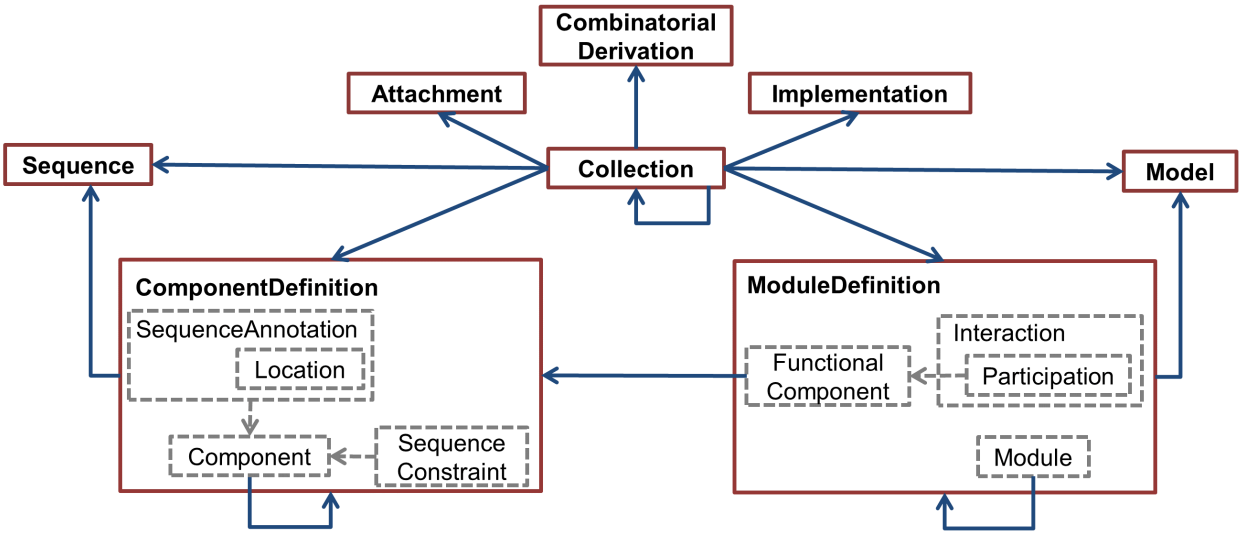


Figure 1: An overview of top level SBOL entities. Solid boxes represent top level entities and solid lines show the relationships. A detailed view is shown for `ComponentDefinition` and `ModuleDefinition`. Dashed boxes and lines show some of the child entities and their relationships respectively.

used as containers to exchange information using SBOL.

SBOL provides several other entities, which are encapsulated within these top level entities, in order to represent complex and hierarchical designs, and to provide additional information. These child entities are also represented as SBOL-OWL classes. For example, a complex genetic part can be represented as a `ComponentDefinition` entity which may include multiple `Component` entities (Figure 6). Whilst a `ComponentDefinition` entity defines a part, a `Component` entity defines the biological usage of a child part in the context of its parent and points to a `ComponentDefinition` with the details of the child part. The latter `ComponentDefinition` may also refer to other child parts. The location of each child is then defined relatively according to its parent using `SequenceAnnotations`. Alternatively, a parent part can simply refer to all of its individual parts without using a hierarchy. The resulting two graphs would be different even though they both refer to the same physical implementation. Other examples include the `SequenceConstraint` term to restrict the relative ordering of subparts and the `MapsTo` term to connect hierarchical design information. Rather than specifying exact locations, `SequenceConstraints` can also be used to derive the

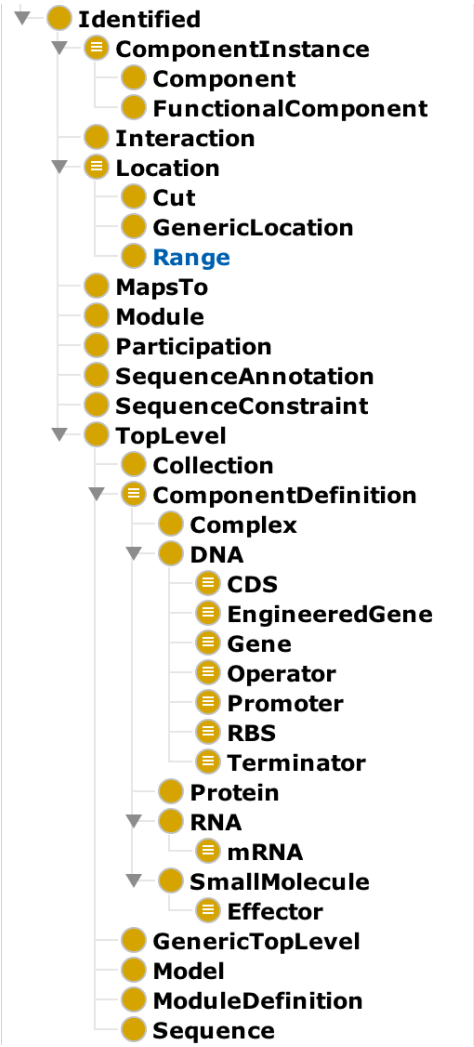


Figure 2: An overview of some of the classes in the SBOL-OWL ontology in Protégé. Terms are created using SBOL entities that are represented in SBOL documents. Both solid yellow icons and yellow icons with three lines represent SBOL-OWL terms. The yellow icons with three lines denote classes that are defined as expressions of other classes.

ordering of parts for a genetic circuit.

Encoding Abstract Entities. SBOL-OWL includes additional terms that are not serialized as SBOL entities. Some of these entities are simply used to classify other SBOL entities and would be useful when querying genetic design information. However, such queries are currently not possible since the exact graph representation of data do not include any reference to these entities. For example, `TopLevel` is not a specific SBOL entity that would be present when designs are serialized. Unified Modeling Language (UML)[32] diagrams in

the SBOL specification[26] often utilize such entities when describing entities with common properties. These interface classes are useful for semantic reasoning and to simplify queries, and hence are included as classes in SBOL-OWL. For example, any of the top level terms such as `Model` or `Sequence` can be referred to as a `TopLevel` entity. Moreover, the modelling of these interface classes in the ontology simplifies the definition of subclasses. Hence, such subclasses' definition comes from their properties and their relationships to other terms but also from their parent terms.

Other examples of such interface classes include `Identified`, which is defined to be any SBOL entity that can be identified uniquely using URIs. The use of interface classes in SBOL-OWL also allows for identifying other entities through *is_a* and *subclass* relationships. For example, `ComponentInstance` in SBOL describes how design elements can be defined using inputs and outputs to create hierarchical designs and modules. However, the `ComponentInstance` entity is not used in serialization. Instead, `FunctionalComponent` and `Component` entities in SBOL are derived from `ComponentInstance`, and are respectively used to serialize information at the structural DNA level to describe sequence composition of parts and at the functional level to describe molecular interactions. In SBOL-OWL, these child entities are represented as classes that are subclasses of `ComponentInstance`, which can therefore be used in queries.

Representing Relationships Between Design Components. One of the reasons to develop SBOL-OWL is to validate SBOL documents using an ontological approach, where logical axioms are used to confirm that the ontology is consistent when it is merged with SBOL documents. This semantic layer provided by the ontology helps identifying inconsistencies through existing reasoners. To enable this approach, it is crucial to capture complex relationships between different SBOL entities ontologically. In addition to modeling the SBOL data model as an ontology, we developed SBOL-OWL to also capture complex validation rules in SBOL, as a set of logical axioms to prevent inconsistencies between the SBOL data model and genetic circuit designs.

Based on the serialization examples in the SBOL specification, relationships between SBOL entities are mainly modeled as *object* properties in SBOL-OWL. These object properties are defined with the `owl:domain` and `owl:range` properties respectively to indicate which SBOL entities would have these properties and which SBOL entities these properties would point to in order to link different SBOL entities. Some of the properties of SBOL entities can be literal values such as strings or numerical values. Such properties linking SBOL entities to literals are modeled as *data* properties in SBOL-OWL. However, if the value is a URI, such as an identifier to an external ontology term, the property is modeled as an object property too.

SBOL specification enforces strict rules to specify the cardinality of properties and whether they are required or not. The following rules were applied to represent these complex relationships between entities and values using SBOL-OWL properties, where possible. These rules can further be applied to extend SBOL-OWL consistently for future versions of SBOL.

- *Exactly one value*: The *domain* and *range* restrictions of the property are defined to capture the relationships between an entity and a value. The property is then defined to be *functional* to indicate that the entity can have at most one value. The class representation of the entity is then restricted to have at least one use of the property through the *some* (*someValuesFrom*) constraint. For example, a `Participation` entity in SBOL must have a `participant` property and the value must be an instance of `FunctionalComponent`. In SBOL-OWL, the `participant` property is defined to be functional so that there can be at most one `FunctionalComponent` for a `Participation` entity. Moreover, the '`participant some FunctionalComponent`' restriction indicates that there must be at least one `FunctionalComponent` for a `Participation` entity. Finally, the `participation` property's *domain* and *range* restrictions point to `Participation` and `FunctionalComponent` classes respectively.

- *Zero or one value*: Similar to the case above, the property is defined to be *functional*,

and *domain* and *range* restrictions are defined to link entities and values. For example, the `version` property of the `Identified` is an optional `String` and can have at most one value. Therefore, this property is defined to be a *functional* data type property. The *domain* and *range* restrictions point to the `Identified` class and `xsd:string` respectively.

- *zero or more value*: The property's *domain* and *range* restrictions are defined to link entities and values. For example, a `ComponentDefinition` entity may have zero or more `SequenceConstraint` entities. Therefore, the `sequenceConstraint` object property's *domain* and *range* restrictions point to `ComponentDefinition` and `SequenceConstraint` classes respectively.

- *one or more*: The *some* restriction on the entity is used to indicate at least one relationship. The property's *domain* and *range* restrictions point to the entity and the value, respectively. However, the property is not defined to be *functional*. An example is the modeling of the relationship between `SequenceAnnotation`s and `Location`s. Each `SequenceAnnotation` can have at least one `Location` via the '`location some Location`' restriction.

In addition to these rules, additional logical axioms are used to represent more complex validation rules where possible. These axioms are included as subclasses that restrict the definition of SBOL entities. For example, when describing hierarchical designs and connecting inputs and outputs of different design components, the component referred to as *"the remote"* must have the `access` type set to `public`. In SBOL-OWL, this constraint is split into logical axioms. Since the `remote` property is required, it is defined as *functional*. Using the existential *some* relationship, "`remote some ComponentInstance`" axiom is defined. Finally, to restrict the potential values of the `access` property for `ComponentInstance`s, we use the clojure axiom `"remote only (access some public)"`.

To increase the flexibility of querying mechanisms using SBOL-OWL, we also defined

inverse properties. For example, in addition to linking a `ComponentDefinition` and a `Component` entity using the `component` property, `isComponentOf` property is also defined, as an inverse property of the former. These inverse properties make the construction of logical axioms easier.

SBOL uses references to many other ontologies to provide the meaning of design entities where possible. Some of these external terms are also included in SBOL-OWL. For example, BioPAX terms `Complex`, `DnaRegion`, `RnaRegion`, `Protein` and `SmallMolecule` are used to indicate types of design components. In addition, Sequence Ontology[33] terms indicate the role of DNA-based components in designs. Similarly, EDAM[34] terms indicate types of external documents that are referred to and the Systems Biology Ontology (SBO)[35] terms are mainly used to classify biological interactions.

The SBOL Vocabulary. Not everything that is necessary to define genetic circuits exists in other ontologies. Therefore, the SBOL standard has been extended with specific terms where necessary. The SBOL specification describes these values as URI constants. For example, the `direction` and `access` properties are useful to, respectively, describe the direction of inputs and outputs, and whether they can be accessed by other designs. In SBOL-OWL, each URI is represented as a class to facilitate the execution of semantic queries. A parent class is also created for a set of related terms that are potential values for a specific SBOL property (Figure 4). For example, `private` and `public` are used as values of the `sbol:access` property when serializing SBOL documents. In SBOL-OWL, `public` and `private` classes are subclasses of the `Access` class. The `access` object property is created in such a way that only entities deriving from `ComponentInstance` can have this property and the *range* is strictly restricted to subclasses of the `Access` class (Figure 3). Similarly, we defined the `Direction`, `Orientation`, `Refinement`, `Restriction`, and `RoleIntegration` classes, each of which has subclasses that are used to describe genetic circuits.

Adding Metadata Classes. SBOL is quite a verbose language. Although the data model is very flexible, it may require several statements to describe a biological concept. For

```
ObjectProperty: sbol:access
    Annotations:
        rdfs:label "access"@en,
        rdfs:comment "The access property is a REQUIRED URI that indicates ..."@en
    Domain:
        sbol:ComponentInstance
    Range:
        sbol:Access

Class: sbol:Access
    Annotations:
        rdfs:label "Access"@en,
        rdfs:comment "Not represented in SBOL directly. It is used in the OWL representation
            to enforce choosing an access type using one of its subclasses."@en
    EquivalentTo:
        sbol:private or sbol:public
    SubClassOf:
        sbol:SBOLVocabulary
```

Figure 3: The definition of the `access` property and the `Access` class. Definitions are shown using the OWL Manchester syntax.[36]

example, a promoter term is represented using SBOL's generic `ComponentDefinition` term. As a best practice, this entity has the value of `DnaRegion` from the BioPAX ontology for the `type` property, and the value of `SO_0000167` (promoter term) from the Sequence Ontology for the `role` property. The situation makes writing complex queries even more challenging when such entities are queried in the context of different information. The minimum information required to describe a promoter part can be simply represented as a single logical axiom, such that an entity is a "Promoter". To facilitate the creation of such logical axioms, SBOL-OWL defines metadata classes (Figure 5A). For example, the `Promoter` metadata class would therefore indicate that any entity belonging to this class would have the `type` property set to `biopax:DnaRegion`, and the value set to `so:SO_0000167` (Figure 5B). Such metadata classes remove redundancies in representing data and enables writing simpler queries.

Metadata classes that are defined as subclasses of SBOL's `ComponentDefinition` include `Complex`, `DNA`, `RNA`, `Protein`, and `SmallMolecule`. These metadata classes correspond to types of `ComponentDefinition` entities specified in the SBOL specification. Each of these classes represent a different type of a design component which can have sequence information in a specified format. SBOL-OWL also enforces the types of sequences that can be associated with each type. Furthermore, we defined `CDS`, `EngineeredGene`, `Gene`, `Operator`, `Promoter`, `RBS`, and `Terminator` classes as subclasses of the `DNA` class to implement SBOL's

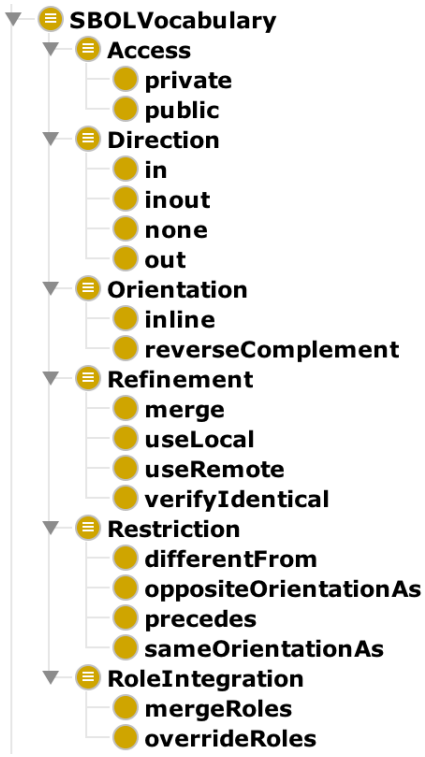


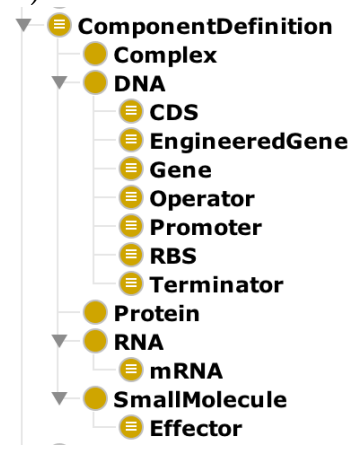
Figure 4: SBOL terms specific to the serialization of genetic circuit designs are described as OWL classes.

best practices.

Querying using SBOL-OWL. SBOL data typically include information about design components that are considered as individuals in SBOL-OWL. These individuals belong to classes in the ontology. Querying in SBOL-OWL is therefore achieved through instance-level inferencing by using complex relationships and restrictions between these classes.

In SBOL-OWL, as in OWL more generally, one queries the data by creating a formal class definition, which then allows standard OWL inference engines[28–30] to determine membership into this class, thereby providing an answer to the query. For example, to find all instances of promoters, one would define the class `Promoter` exactly as in Figure 5B, and ask for all members (or instances) of this class. As described in the previous section, SBOL-OWL includes a number of these common queries, built-in as metadata classes. Figure 6C provides a slightly more complex query example, asking for all DNA components that have some child components.

A)



B)

```
Class: sbol:Promoter
    Annotations:
        rdfs:label "Promoter"@en,
        rdfs:comment "Promoter DNA component"@en,
    EquivalentTo:
        sbol:role some <http://identifiers.org/so/SO:0000167>
    SubClassOf:
        sbol:DNA

Class: sbol:DNA
    Annotations:
        rdfs:comment "DNA component definition"@en,
        rdfs:label "DNA"@en
    SubClassOf:
        sbol:ComponentDefinition,
        sbol:type some <http://www.biopax.org/release/biopax-level3.owl#DnaRegion>
```

Figure 5: A. Metadata classes facilitate the creation of logical axioms. These classes often have human readable names and therefore queries are relatively easier to write compared to using identifiers from external ontologies and linking these identifiers to other SBOL specific terminology. B. Metadata classes include `Promoter` and `DNA`, which are shown using the OWL Manchester syntax. The `Promoter` class is defined to represent all entities that have the `sbol:role` property set to the `SO:0000167` promoter term from SO. It is a subclass of the `DNA` class that represent DNA entities. These entities are of type `ComponentDefinition` and have the `sbol:type` property is set to `biopax:DnaRegion`.

As shown in Figure 6C, because queries are themselves classes, they can easily be reused in different queries. This is exactly the motivation by pre-creating metadata classes such as `DNA` and `Promotor` (Figure 5B). Moreover, a query class can refer to itself, thereby creating a recursive query, which are ideal for hierarchical designs such as are often used in SBOL. In contrast, graph-based SPARQL queries do not allow for recursion, and thus could only retrieve such information via multiple programmatic calls.

The challenge of querying SBOL hierarchical designs is one of our key motivations behind

the design of SBOL-OWL. In SBOL, the relationship between a child and a parent part is not direct. Further, a part can be used in several different designs, and its use may have different properties. Thus, the class `ComponentDefinition` in SBOL refers to a part, whilst a `Component` refers to the use of that part by its parent (linked by the `component` property). These complex part-whole relationships become especially tedious to query as the size and complexity of designs increase. SBOL-OWL provides an ontological solution to query these complex designs as shown next.

Semantic Reasoning for Genetic Circuit Designs. The SBOL-OWL ontology is useful not only to formalize the standard representation of genetic circuit designs, but also to exploit the use of description logics to infer information about designs. Here, the ontology is used to achieve semantic reasoning. Typical competency questions to measure the validity of SBOL-OWL involve queries to extract information based on common properties of design entities. A simple example would be to retrieve all DNA parts which have subparts. A more formal description regarding this competency question is to return a list of `ComponentDefinition`s that are of type `DnaRegion` and that have `some Component`s. Consider the design in Figure 6 from the Registry of Standard Biological Parts[10] (http://parts.igem.org). The design starts with the BBa_F2620 cell-cell communication receiver, which was developed by Canton and colleagues to standardise the characterization of genetic parts.[37] BBa_F2620, also known as the PoPS (polymerase per second) receiver, includes a promoter at the end, which is activated upon sensing a signalling molecule. In this example, the receiver is followed by a RBS and a CDS encoding for GFP in order to report the sensing of the signalling molecule. The terminator, which is included in BBa_F2620 consists of two individual terminator parts, and hence it is a double terminator. Therefore, the query should return the entities for the design, the PoPS receiver device and the double terminator.

Queries in SBOL-OWL typically utilize the *some* (*SomeValuesFrom*) restriction, since it is commonly used to capture relationships between SBOL entities. A *some* restriction

indicates that there is at least one relationship and does not rule out other possibilities.[38]

The OWL statement in Figure 6C represents SBOL entities that act as DNA-based composite parts, that are formed of other parts (example data: Table 2.3). This statement can also be used as a query to classify all SBOL entities conforming to this description. As a result, this query is used to retrieve part definitions representing DNA and having some parts. The query in Figure 6C is further simplified via the use of domain specific metadata classes and becomes "`DNA and (component some Component)`".

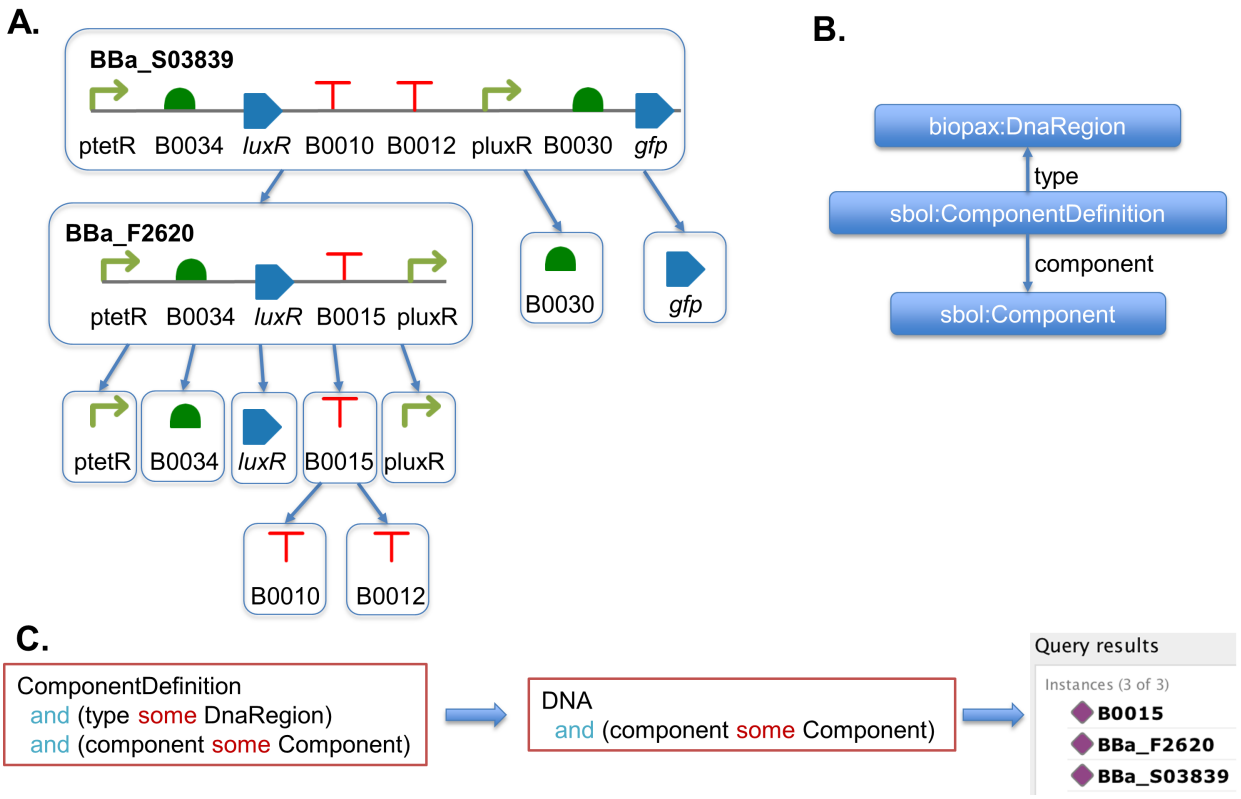


Figure 6: A. A hierarchical genetic circuit design example. Each box represents a genetic part or a design. Arrows display the relationships between parent and child parts. B. SBOL data model to represent part-subpart relationships for DNA-based parts. Parts are represented using the `ComponentDefinition` entity. A `Component` entity stores the reference to the child part and is connected to parent part via the `component` property. Arrows indicate how three different entities are related in the SBOL data model. C. SBOL-OWL based ontological queries (example data: Table 2.3).

The use of these metadata classes is especially important in more complex queries. These classes provide a way of semantic collapsing to create simpler queries. A competency question

to retrieve parts which contain promoters can be formulated as follows: How do I return `ComponentDefinition`s that have `some Components`, which are `Promoter`s and are of `type DnaRegion`? This question can be captured in the form of a SPARQL query which is a graph pattern. An example SPARQL query is shown in Figure 7A. The query is constructed based on the SBOL data model, a subset of which is shown in Figure 7B. As it can be seen, SPARQL queries use exact graph relationships in order to report matching subgraphs in SBOL data. Even for this example, SPARQL queries can become quite complex using graph-based search mechanisms (Figure 7A-B). Such queries can be more easily constructed using SBOL-OWL (Figure 7C, example data: Table 2.3). Due to the way information is structured in SBOL-OWL queries are semantic. Additional `DNA` and `Promoter` classes are used to introduce domain knowledge without making changes in SBOL data. This semantic collapsing enables writing simpler and shorter queries via commonly used terms used in the synthetic biology domain.

Considering the design in Figure 6A, neither the SPARQL nor the SBOL-OWL based query would list the BBa_S03839 since this design does not include a promoter directly. However, the PoPS receiver device would be listed since it contains two promoters as first level parts. Clearly, this situation is not ideal and particularly a problem when using SPARQL queries.

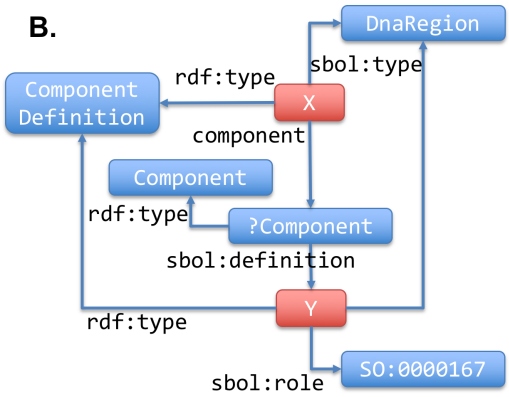
SBOL-OWL terms and the concept of semantic collapsing can be used to create recursive queries to infer all of the required information. These queries can be constructed based on the transitivity of relationships using several classes and properties. This type of queries often involves querying based on a particular design component.

Recursive queries are especially needed when querying the ordering of biological parts. The location of a part can be crucial for the functioning of a genetic circuit. These locations affect *cis* interactions, which are about structural organization of DNA sequences. For example, for a genetic circuit to function as predicted, a promoter comes before a RBS to drive the expression of RNA molecules, and a RBS comes before a CDS to initiate translation.

A.

```
PREFIX sbol: <http://sbols.org/v2#>
PREFIX biopax: <...>
PREFIX so: <...>
SELECT ?X
WHERE
{
 ?X a sbol:ComponentDefinition ;
    sbol:component ?Component ;
    sbol:type biopax:DnaRegion .
  ?Component a sbol:Component ;
    sbol:definition ?Y .
  ?Y a sbol:ComponentDefinition ;
    sbol:type biopax:DnaRegion ;
    sbol:role so:SO:0000167 .
  }
```

B.



C.

```
DNA and (component some (definition some Promoter))
```

Figure 7: A. A SPARQL query to find parts that contain promoters. The query describes a graph pattern which is then used to search for matching design information. Parts and promoter subparts are represented with X and Y variables respectively. B. Graphical representation of the SPARQL query in A. C. The SBOL-OWL based representation of the SPARQL query shown in A. In this case, the query is created using the OWL syntax and logical axioms. Any `DNA` part that has `at least one component`, which is `defined` to be a `Promoter` is inferred from the design information. This query is captured as an OWL class which is then used by reasoners to classify parts. The result of the query is the list of individuals matching the class definition (example data: Table 2.3).

Positional constraints also affect the final concentrations and localizations of gene products and hence affect *trans* interactions. Therefore, it is important to gather information about a part with reference to other parts' locations.

SBOL provides `SequenceConstraint` entities to represent relative ordering information between any two parts. For example, in order to represent the ordering of all parts in the PoPS receiver device, defining four `precedes` pairwise constraints is sufficient (Figure 8). Each constraint indicates which part precedes the other in the design.

Finding the relationships based on the connection of nodes and edges using direct neighborhood is relatively easy and therefore parts proceeding others can be retrieved using SBOL-OWL based queries. Figure 9A displays such a query to find the part that comes after the ptetR promoter. Here, we use the `value` (*hasValue*)[12] restriction in order to reference genetic parts in queries. However, it is not straightforward to find all parts that come after

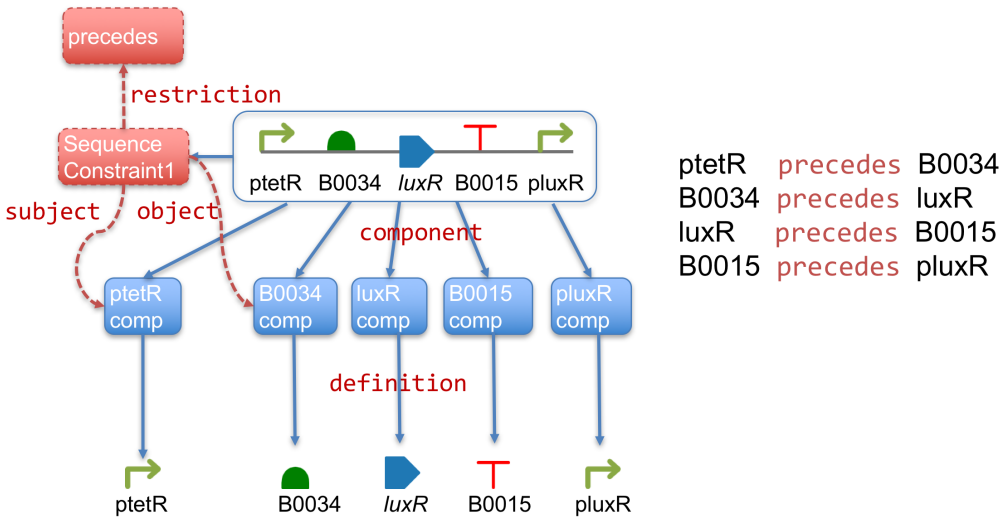


Figure 8: The PoPS receiver device consists of 5 parts. 4 pairwise precedes relationships are sufficient to represent the order of these parts. In the figure `sequenceConstraint1` is used to represent the relation between ptetR promoter and the B0034 RBS parts.

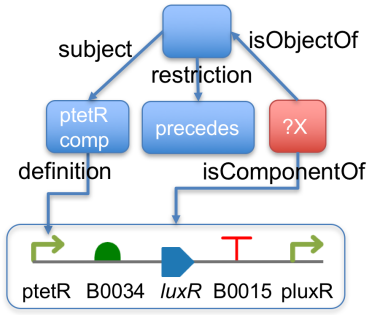
the ptetR part. Parts can be grouped semantically to indicate that any part coming after ptetR should be recursively queried. The latter approach is demonstrated in Figure 9C. The `ptetRFollower` query is defined as an OWL class and its definition refers back to itself. When the class is submitted to reasoners, all parts that come after ptetR are considered. As a result, the query returns components for B0034, luxR, B0015 and pluxR.

Even this approach may not be sufficient when parts are included in designs that include also other parts. Consider the design in Figure 6A. The query in Figure 9C would not be able to retrieve the B0030 RBS and gfp CDS parts, since they are not included in the parent design of ptetR. The solution to this problem is to update SBOL-OWL based queries and to incorporate parent designs in queries. Such an approach is invaluable to query hierarchical designs and to find uses of a single part. An example of finding the use of a part in all designs is shown in Figure 10. The `ptetRParent` class in Figure 10A is used as a query to find all uses of the ptetR parents and their use recursively. The query in Figure 9C is then updated to also refer to parents of the ptetR part. As a result, the query in Figure 10B returns all parts that come after ptetR, but also any that follow its direct or indirect parents recursively. The query result is shown in Figure 10C.

A)

```
Component and (isComponentOf value BBa_F2620) and
  (isObjectOf some (
      (restriction value precedes)
      and (subject some (definition value ptetR))))
```

B)



C)

```
Class: ptetRFollower
  Component and (isComponentOf value BBa_F2620) and
    (isObjectOf some (
      (restriction value precedes)
         and( (subject some (definition value ptetR))
              or (subject some ptetRFollower))))
```

D)

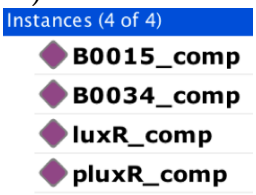


Figure 9: A. SBOL-OWL based query to find the part that comes after the ptetR part. B. Graphical representation of the same query (Table 2.13). C. Recursive version of the query which is defined as a class to return all parts that come after ptetR (Table 2.13). D. Results of the recursive query in C.

Hierarchical designs can be especially complex for querying when designs are split into functional modules, each of which may include information about molecular interactions between biological parts. These functional layers are represented using `ModuleDefinition` entities in SBOL. Each `ModuleDefinition` can include a set of `Interaction` entities and references to other child `ModuleDefinition` entities through `Module` entities (see Figure 1). These `Module` entities are used to provide mappings between components of parent and child designs. Although hierarchical designs facilitate reuse of information, these designs

A)

```
Class: ptetRParent
  DNA
    and ((component some (definition some ptetRParent))
          or (component some (definition value ptetR)))
```

B)

```
Class: ptetRFollower
Component and (isComponentOf some BBa_S03839_node) and
  (isObjectOf some (
    (restriction value precedes) and
      ((subject some (definition value ptetR))
        or (subject some ptetRFollower)
        or (subject some (definition some ptetRParent)))))
```

C)

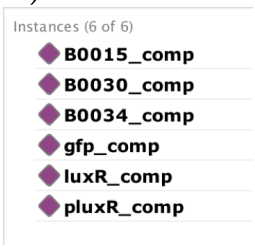


Figure 10: A. When ptetRFollower is submitted to reasoners, both BBa_F2620 and BBa_S03839 are inferred as designs where ptetR is used (Table 2.13). B. The ptetRFollower class is updated to include designs where ptetR is used in order to recursively incorporate information about parent designs into the query (Table 2.13). C. Results show parts that come after the PoPS receiver device (BBa_F2620) too.

may need to be flattened to increase the understanding of relationships of SBOL entities, for example to generate computational models or to visualise designs. Figure 11 shows a hierarchical SBOL design using eleven `ModuleDefinitions`. The 'circuit_0x78 environment' `ModuleDefinition` entity describes the overall design using three different inputs and an output. The relationships between the output and the inputs are described using the 'circuit_0x78' `ModuleDefinition` entity. This child entity includes information about the binding of inputs to different protein molecules. Information about the production of each type of protein molecules is encapsulated in a different `Module` entity and is referred to by the 'circuit_0x78' `ModuleDefinition` entity. Therefore, the querying of interactions requires the consideration of information at different levels. This circuit can be queried semantically as shown in Figure 12 (example data: Table 2.5). All child `ModuleDefinitions` are classified

using the `circuit_0x78_Module` class, and direct interactions belonging to any entity in this set is reported as interactions of the genetic circuit.

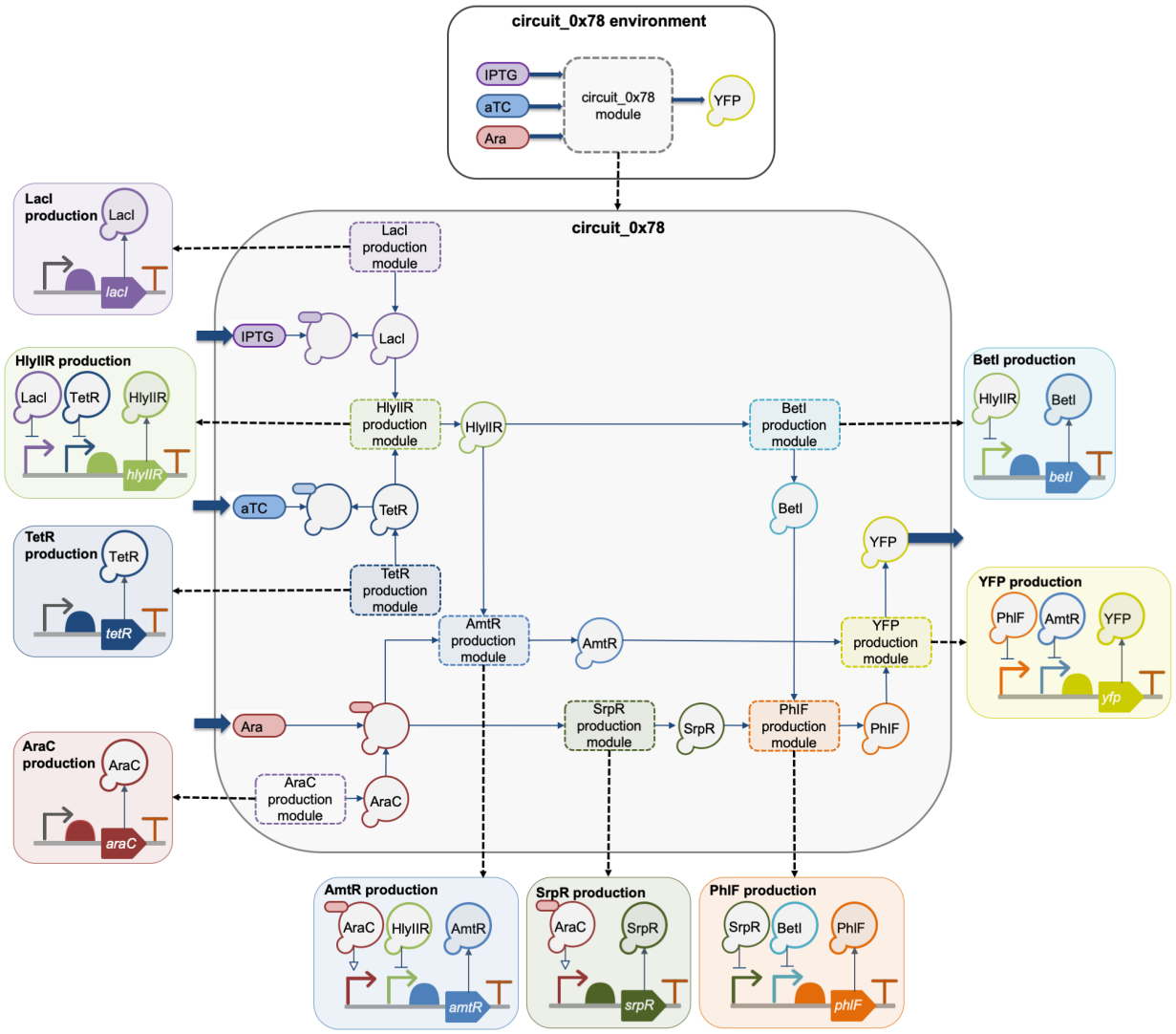


Figure 11: The 'circuit_0x78_environment' genetic circuit design (adapted from[39]), with three environmental inputs and one output, is shown using SBOL Visual glyphs[40] (http://sbolstandard.org/visual). This entity includes the 'circuit_0x78' design, which further includes nine sub designs, all of which include information about different molecular interactions. Designs are represented as `ModuleDefinition`s and their use in parent designs are represented as `Module`s. For example, the 'circuit_0x78_environment' `ModuleDefinition` includes 'circuit_0x78 module' which refers to the 'circuit_0x78' `ModuleDefinition`. `ModuleDefinition`s and `Module`s are shown with rounded rectangles using solid lines and dashed lines respectively. Dashed arrows show the connections between these `ModuleDefinition`s and `Module`s.

Validating Genetic Circuits. Validating SBOL data can be challenging. Although,

```
Class: circuit_0x78_Module
ModuleDefinition and
  (isDefinitionOf some
    (Module and (
      (isModuleOf some circuit_0x78_Module)
      or
      (isModuleOf value circuit_0x78_environment_md))
    )
  )

Class: circuit_0x78_Interaction:
Interaction and (isInteractionOf some circuit_0x78_Module)
```

Figure 12: The `circuit_0x78_Module` class is used to classify all child ModuleDefinitions of 'circuit_0x78_environment_md' (Table 2.5). The `circuit_0x78_Interaction` class then reports 34 interactions in these submodules.

SBOL is serialised as RDF/XML, RDF graphs containing SBOL data may not always be SBOL compliant. This is due to SBOL adopting a special nesting to support XML tools and embedding all child entities with top level entities. As a result, the serialisation of an RDF graph, in addition to the relationships of graph nodes, affects whether data are SBOL compliant or not. However, SBOL data may be stored in RDF triplestores. When SBOL graphs are retrieved from these triplestores, they are transformed using the specific SBOL serialisation prior to validating data.

SBOL-OWL provides a complementary approach to validate SBOL data. Since the SBOL-OWL ontology captures constraints between SBOL entities, it acts as a schema for SBOL graphs, whether these graphs are generated by using SBOL libraries or downloaded from RDF repositories. These constraints are available computationally and can be used by tools to validate SBOL graphs.

SBOL-OWL can also be used directly for validation via reasoners. Since the constraints are available as ontological axioms, reasoners can be executed to check whether SBOL data are consistent or not, without writing any queries. SBOL-OWL captures how entities are related to each other using `domain` and `range` properties. As a result, the incorrect use of a property would be automatically reported by a reasoner. For example, the sbol-11608[26] (Table 1) validation rule specifies that a `ModuleDefinition` entity can have a `model` property which points to `Model` entities. The example SBOL file in Table 2.11 includes the CRISPR_Template `ModuleDefinition` entity. The file is reported to be inconsistent by a

reasoner since the `model` property of this entity points to a `Sequence` entity. Since the `model` property's *range* property is defined to be `Model` which is disjoint with `Sequence` (that is an entity cannot both be an instance of `Model` and `Sequence` entities), reasoners can infer that the file is not a valid SBOL file.

Another example is the validation of the sbol-10503[26] (Table 1) SBOL validation rule which specifies that a single `ComponentDefinition` entity cannot have conflicting part types. For example, it cannot be used to represent both a DNA and protein part. Since SBOL-OWL describes different part types as disjoint, an SBOL entity cannot be of both types. The example SBOL file in Table 2.6 contains the 'EYFP_cds' entity that is defined to be both DNA and RNA. As a result, the reasoner reports the file as inconsistent. Additional examples of invalid SBOL data are presented in Table 1.

We then applied SBOL-OWL to test the validity of designs in SynBioHub repository. Using an automated approach (Table 2.14), 372 designs (Table 2.12) from the '2018 iGEM Distribution Plate 1' collection were initially downloaded from synbiohub.org, merged with SBOL-OWL and submitted to reasoners to check their validity using the semanticSBOL library that has been developed as part of this work. All of these designs were confirmed to be consistent using SBOL-OWL.

Although, here, we used reasoners to automate validating SBOL graphs according to ontological axioms, this automation approach has limitations. Even though all 372 designs were confirmed to be consistent, there may be cases that the reasoners cannot identify inconsistencies. This issue is due to the open-world assumption used by reasoners. Regarding the inconsistent SBOL example in Table 1, sbol-11608, in which a `ModuleDefinition` entity includes the `model` property that refers to a `Sequence` entity rather than a `Model` entity, reasoners could identify the inconsistency. In this example, necessary and sufficient conditions are met, since the `model` property can refer to a `Model` entity which is defined to be disjoint from `Sequence`. However, SBOL-OWL would not find inconsistencies when a required property is missing. For example, the validation rule sbol-10402 - *"The elements property of*

Table 1: Examples of invalid SBOL data. The table lists some of the SBOL validation rules[26] that can be checked with SBOL-OWL. The thirds column includes references to SBOL files with inconsistent information and explains why these files are inconsistent.

Rule	Description	Example
sbol-10503	`ComponentDefinition` entities cannot have conflicting part types.	Table 2.6: The 'EYFP_cds' entity is defined to be both DNA and RNA.
sbol-10511	`ComponentDefinition` entities should have biological roles compatible with their types.	Table 2.7: A protein part is defined to have a `promoter` role, which can only be used for DNA parts.
sbol-10516	`ComponentDefinition` entities should have sequences compatible with their types.	Table 2.8: The same sequence was used for different CDS and terminator parts.
sbol-10807	The `ComponentInstance` referred to by the `remote` property of a `MapsTo` MUST have an `access` property that contains the URI http://sbols.org/v2#public.	Table 2.9: The 'cas9m_BFP' `MapsTo` entity's `remote` property refers to a `FunctionalComponent` that is defined to be `private`. The `access` property of this `FunctionalComponent` entity must be `public`.
sbol-11406	The `object` property of a `SequenceConstraint` MUST NOT refer to the same `Component` as the `subject` property of the `SequenceConstraint`.	Table 2.10: The 'CRP_b' is referred to as both an `object` and a `subject` for a `SequenceConstraint` entity. These properties must refer to different `Component` entities.
sbol-11608	Each URI contained by the `models` property of a `ModuleDefinition` MUST refer to a `Model`.	Table 2.11: The 'CRISPR_Template' `ModuleDefinition` entity's `model` property refers to a `Sequence` entity. It must point to a `Model` entity.

a ***`Sequence`*** *is REQUIRED and MUST contain a String."* cannot be checked with SBOL-OWL. When information is missing, reasoners assume that this information is unknown and unless specified there may be additional information. Even though, the `Sequence` entity is defined in SBOL-OWL to have at least one `elements` property, necessary and sufficient conditions are not met to report the inconsistency. On the other hand, if the `elements` property is included more than once, reasoners can report this inconsistency since this property is defined to be functional in SBOL-OWL and hence cannot be included more than once. In summary, reasoner-based validations can be used when all required SBOL properties are

provided and SBOL specific values are chosen from the SBOL data model.

semanticSBOL. We developed the semanticSBOL library to facilitate the application of reasoners to SBOL graphs. The library includes methods to combine SBOL graphs with SBOL-OWL and the resulting graphs can directly be submitted to reasoners, either manually or programmatically. The library includes basic methods to check the consistency of SBOL graphs, execute OWL-based semantic queries in the form of ontology classes, list SBOL entities that are classified according to these class definitions, and add new properties to SBOL entities (see Table 2.15 for an example). The default SBOL namespace is used to resolve full URIs of SBOL entities used in these queries. Custom entities can also be incorporated into the queries by using the format '*prefix:entityName*', where `prefix` represents an abbreviation for a namespace and `entityName` represents the name of a custom entity. Prefix-namespace declarations can be passed to semanticSBOL as a parameter. We used this approach for the '2018 iGEM Distribution Plate 1' collection, to verify that designs annotated to be specifically for *Bacillus subtilis* do not contain an *Escherichia coli* promoter (Table 2.14).

Discussion. SBOL-OWL has been developed to complement the standardization efforts in synthetic biology. So far, the community has focused upon a common syntax to facilitate the reproducibility of designs. Our goal here is to make information about genetic circuits machine understandable via an additional semantic layer. SBOL-OWL captures the SBOL specification as a Semantic Web ontology, standardizing both the syntax and semantics of the SBOL data model in a computationally tractable format. Semantic representation of genetic circuits has significant advantages and the potential to create new applications for synthetic biology. Examples include creating applications for data integration, visualization, model annotation and automated reasoning.

SBOL-OWL provides a machine-accessible schema for SBOL graphs by unifying different serialisation rules, which are available in the SBOL specification[26] in the form of free text and multiple tables. Although, the specification also includes several UML diagrams to explain

the relationships between different SBOL entities, there is no one-to-one mapping between these diagrams, and how SBOL entities and their relationships are serialised. For example, a UML diagram indicates that a `Collection` entity is linked to a `TopLevel` entity via the zero-to-many 'members' property, which in fact is serialised using multiple `member` properties. SBOL-OWL explicitly defines how each property should be correctly linked to different entities, providing a specification for the serialisation of SBOL graphs. In the ontology, some of these serialisation rules are defined via *domain* and *range* properties indicating how an SBOL property can be used to connect two SBOL entities. Functional properties constrain the unique use of SBOL properties, and existential restrictions specify required properties for SBOL entities. In the Semantic Web stack, OWL sits on top of RDF. Similarly, SBOL-OWL is directly built upon SBOL and can be more easily integrated with SBOL graphs compared to UML diagrams. Since SBOL-OWL is also available as an RDF graph, the merge of SBOL-OWL with an SBOL graph results with another SBOL compliant graph that can be processed by tools, for example to store in SBOL repositories.

As the rules governing the serialisation of SBOL graphs can be programmatically checked by tools, SBOL-OWL can also facilitate the use of off-the-shelf reasoners to verify genetic circuits and to report inconsistencies in an automated manner. In such a validation process, reasoners use logical axioms provided by SBOL-OWL to validate SBOL graphs. This approach can be used together with existing programmatic solutions, to delegate some of the validation related checks to reasoners. Moreover, this approach is useful when SBOL documents are processed as graphs without the help of a programmatic library. Currently, SBOL graphs can also be validated by executing SPARQL queries to check for incorrect uses of SBOL entities. Although a SPARQL query can be used to report the incorrect use of a property (Supplementary Table 1), such an approach would require executing several similar queries to check for correct usages of different SBOL properties. On the other hand, since the SBOL data model is captured in the form logical axioms, reasoners can use these axioms to computationally identify inconsistencies without a need to execute any query.

Another advantage of using logical axioms to capture the SBOL data model is the ability to write semantic queries. This approach allows writing queries based on semantic relationships of SBOL entities rather than using neighbourhoods of entities in SBOL graphs. An example query is shown in Figure 6C to retrieve composite parts that are formed of other parts. Although a SPARQL query can be used to retrieve the same information (Supplementary Table 2), these queries may become more complex as the number of edges referred to increases (Supplementary Table 3).

The use of SBOL-OWL queries rather than complex patterns of graph-structure based SPARQL queries can be more intuitive and can be written relatively more easily (Figure 7). Moreover, in some cases, inferencing tasks may be difficult to implement using SPARQL alone. For example, multiple SPARQL queries may need to be executed, requiring the integration of results programmatically (Supplementary Table 4 & 5). On the other hand, multiple and complex graph queries which may be due to hierarchical representation of biological design information can be represented using relatively simpler ontological queries (Figure 9, 10 & 12).

Semantic collapsing techniques can be exploited through metadata classes that are introduced in SBOL-OWL for commonly used design concepts. This approach makes the SBOL data model easier to understand and facilitates writing simpler queries. For example, a complex SBOL representation for a design component with the DNA type and a promoter role is simply represented via the `Promoter` class in SBOL-OWL. Moreover, hierarchical queries are simplified via class-subclass relationships. Child entities implicitly inherit properties of their parents. This approach can remove redundancies in queries and can also be adopted for data representation and visualization using SBOL in the future.

Due to capturing parent-child relationships, SBOL-OWL may allow writing queries that are not currently possible to execute. Abstract SBOL entities, such as `TopLevel` and `ComponentInstance`, are not included when genetic design information is serialized. Instead, entities such as `Sequence` and `ComponentDefinition` that inherit from these abstract

entities are serialised. As a result, abstract SBOL entities cannot be included in standard querying mechanisms. SBOL-OWL exploits the ontological representation of the SBOL specification and enables the use of any SBOL entity in queries. In some cases, a query would not even be needed when using a semantic approach, when inferred knowledge becomes part of the underlying graph. This approach can be used to retrieve all direct and indirect types of an SBOL entity (Supplementary Table 6). Similarly, it would be possible to retrieve all `TopLevel` entities without explicitly defining which SBOL entities derive from `TopLevel`. For example, if a design includes information about a promoter part (represented as a `ComponentDefinition` entity) and its sequence (represented as a `Sequence` entity), then both entities are also inferred to be `TopLevel`, since both `ComponentDefinition` and `Sequence` are subclasses of `TopLevel` (Supplementary Table 7).

Representing genetic designs using RDF graphs already exploits existing Semantic Web tools. Design information can be stored in RDF databases, called triplestores, and standard RDF libraries can be used to execute graph queries. SBOL-OWL adds a semantic layer over these queries. SynBioHub[18,19] has previously successfully demonstrated the creation of instances of these triplestores. SBOL-OWL can potentially facilitate integration of data, distributed over multiple SynBioHub instances using a method called data federation.[41] Using this approach, data are left in remote repositories, and a common semantics is used to integrate data from sub queries. Since SynBioHub is a graph database and stores SBOL data, SBOL-OWL can mediate the data integration process. SBOL-OWL captures labels and descriptions of SBOL entities and therefore can also be used to improve the readability of SBOL graphs and their visualisation. The ontology has been integrated into SynBioHub, which uses SBOL terms to list synthetic biology related information and uses SBOL-OWL to explain the meaning of SBOL specific terms.

Semantic reasoning has potential to verify genetic circuit structures. Currently, constraints between any two DNA parts can be captured using SBOL. Although the terminology to define the relationship is not rich, efforts in this area are ongoing. SBOL-OWL can be

easily integrated with the set of new terms in order to validate genetic circuits based on the order of DNA components.

Although we demonstrated semantic reasoning using OWL-DL, other reasoning methodologies can also be explored. The SPARQL language has recently been extended with entailment features for semantic reasoning using RDFS. However, not all SPARQL tools and triplestores currently support entailment regimes. Here, we could demonstrate this concept using Jena (Table 2.1). SBOL-OWL has been created using OWL semantics, since OWL offers richer features to encode the complex SBOL data model. Since we also provide SBOL-OWL in the RDF format, a subset of SBOL-OWL can be utilised in RDFS reasoning due to the overlapping features of OWL and RDFS to represent some of the relationships between terms, for example using the rdfs:subclass property to link parent and child terms. To demonstrate this approach, we asked the RDFS reasoner to retrieve all parent terms for `ptetR` in Figure 6. Due to defining relationships of SBOL terms in SBOL-OWL, the reasoner could return `ComponentDefinition` as a parent term, but also the `TopLevel` and `Identified` SBOL terms. Therefore, if RDF statements representing SBOL-OWL are uploaded into a triplestore supporting RDFS entailment regimes, the capability of SPARQL querying would improve.

Semantic reasoning using OWL-DL reasoners indeed has limitations. As demonstrated here, OWL-DL reasoners cannot identify all cases of inconsistencies due to adopting open-world assumptions. For example, even though required properties are modelled through the `some` relationships, which indicate necessary conditions to have at least one member, such relationships may not be enough to provide sufficient conditions for reasoners to identify missing properties. Therefore, SBOL-OWL can be used together with existing solutions, for example to substantially reduce the number of SPARQL queries to check the validity of SBOL data or to reduce the number of programmatic operations. Another limitation of these reasoners is that they can classify terms and individuals based on a semantic query, and returns a set of entities rather than returning tuples to provide more detailed information. On

the other hand, SPARQL is powerful to return results in the form of tables in which columns represent variables queries. Moreover, SPARQL queries can be used to return results in the form of graphs. There is also development in combining the querying power of SPARQL with OWL-DL's powerful semantic representation of domain entities.[42] SPARQL-DL,[43] for example, is a subset of SPARQL and has been developed for ontological queries. SBOL-OWL can potentially be used to facilitate the creation of SPARQL-based semantic queries using such tools with SPARQL reasoning capabilities.

As the SBOL specification is developed and new entities are added to cover even more types of biological data, it will be important to keep track of changes computationally, in order to create new libraries or to update existing ones. SBOL-OWL is based on the latest SBOL 2.2.1 specification, which is 132 pages and is the 6th release since the 2.0.0 specification. Having a formal definition of the relationships of SBOL entities may facilitate governing these changes. Having the SBOL specification in the form of a machine accessible ontology will potentially facilitate the development of automated approaches to computationally verify and validate designs. SBOL-OWL already provides restrictions about how SBOL entities can be linked together and can be useful as a first step to verify genetic circuit designs. It can also be used to auto-generate libraries for tool developers.

SBOL-OWL can also facilitate utilizing large amount of biological data that can be mined for design information, through the integration of other ontologies. SBOL has already adopted the Provenance Ontology (PROV-O, `https://www.w3.org/TR/prov-o/`) to provide provenance information about designs. The SyBiOnt[44] has been developed as an application ontology and is promising as a way of providing an integration mechanism to different ontologies. The linking of SBOL-OWL and SyBiOnt can bridge the use of existing biological data and design information to create reliable biological applications. Since RDF is commonly used to capture semantic annotations for computational models in biology, SBOL-OWL will further facilitate the linking of these models and genetic circuit designs.[45–49]

Currently, SBOL-OWL is limited to the type of data that can be captured using SBOL

and how the SBOL specification is developed. In the future, the specification can be tweaked slightly for more appropriate OWL formalization. This approach would simplify the serialization of SBOL into a bag of triples, whose semantics can be controlled by SBOL-OWL. The ontology is affected by some of the issues inherent in SBOL. Some of the properties, such as "`role`", are used to define relationships between more than two different entities and each representation may have different requirements.

SBOL-OWL is a useful resource for the community as a semantic layer for genetic circuit designs. This ontology provides a schema for SBOL graphs which are used by tools to read and write information about genetic circuits. Similarly, we expect that SBOL-OWL will be used by SBOL tools. Developers can use it to create new applications, for example to auto generate libraries, to provide easy-to-use tools for intuitively querying genetic circuits, to integrate data and so on. This ontology can play a key role to integrate huge amount of design information that is already available through the use of linked data and Semantic Web approaches.

Methods

Constructing the Ontology. The SBOL-OWL ontology was programmatically constructed using Tawny-OWL[50], an API that provides high-level access to create entities for an ontology and to define the semantic relationships between those entities. Due to the built-in features of Tawny-OWL, SBOL-OWL has been developed using a textual user interface within an integrated development environment, which allows errors to be be identified when the resulting code is compiled. Moreover, SBOL-OWL utilises ontology patterns provided by Tawny-OWL. These patterns allow utilising ontology modelling solutions with less effort.

Clojure[51] was chosen as the programming language since Tawny-OWL is based on this language. Clojure is a functional programming language and Tawny-OWL exploits the advantage of this language to provide both a custom domain specific language and a set of

macros to create ontologies. Here, additional macros were created in addition to using the Tawny-OWL API when creating the SBOL-OWL ontology. The resulting ontology was exported in OWL, RDF and Manchester syntax[36] formats. The HTML version of the ontology was automatically generated using LODE[52] from the RDF version of SBOL-OWL.

Testing the Ontology with Standard Definition of Genetic Circuits Designs. SBOL-OWL was specifically developed to provide a semantic layer for SBOL and hence we demonstrate how the ontology can be applied to genetic circuit designs that are represented using SBOL. SBOL-OWL does not require any changes to be made in SBOL files. However, a semantic reasoning process requires that information about genetic circuit designs and the ontological information defining semantic relationships between types of entities (or classes) are integrated and presented to reasoners together. To demonstrate this idea, we provide the semanticSBOL Java library to merge the RDF graph content of an SBOL file and the RDF graph representation of the ontology. The resulting graph essentially stores classes from the ontology and individuals that come from the design, and can be submitted to existing reasoners directly either manually or using semanticSBOL programmatically. Merging RDF graphs was carried out using the Jena RDF library (`https://jena.apache.org`). The libSBOLj[53] library was used to handle SBOL specific data programmatically, for example to generate some of the examples.

Automated Reasoning and Querying. Protégé (version 5.2.0) was used to execute semantic queries manually via built-in reasoners. Protégé's HermiT reasoner (version 1.3.8) and DL Query plugins (version 4.0.1) were used to manually execute semantic queries. Protégé's SPARQL plugin (version 2.0.2) was used to manually query for graph patterns. In this work, simple queries were executed directly as Description Logic queries via Protégé, and complex queries were defined as ontology classes. The resources provided here are based on Semantic Web technologies, and tools that support reasoning can be used.

In order to automate the use of reasoners programmatically, we developed the semanticSBOL library. The library was implemented using the Java programming language as a

Maven (`https://maven.apache.org`) project, and is dependent on libSBOLj,[53] Hermit,[28] OWL-API,[54] and Jena libraries. The semanticSBOL library executes queries using the OWL Manchester syntax.[36]

SBOL-OWL enabled RDFS reasoning was demonstrated using Jena programmatically (Table 2.1). The SBOL representation of the example in Figure 6A was used as data and the RDF version of the SBOL-OWL ontology was used as the data schema. Only the statements for the ptetR resource, with the type property, were listed.

Availability. The resulting ontology files and the source code to produce the ontology together with the semanticSBOL Java library and examples demonstrating the application of the ontology for genetic circuit designs are available from `http://sbolstandard.org/ontology`. The source code and examples are hosted in a GitHub repository, available at `https://github.com/dissys/sbol-owl`. The `sbol-owl` subfolder contains the Clojure project that is used to create the SBOL-OWL ontology, the `sbol-sem` subfolder contains the semanticSBOL library and the `examples` subfolder includes example files. Files referred to in this work are listed in Table 2.

Author Contributions. All authors contributed to the manuscript, and to the development of the ontology terms and examples. P.L. and G.M. encoded the ontology programmatically. G.M. developed the semanticSBOL library. J.H.G., P.L., and G.M. constructed semantic queries. J.A.M., A.W., C.M. and G.M. integrated the ontology into the sbolstandard.org and provided support to fetch data from SynBioHub.

Acknowledgement

Authors thank Jacob Beal and Richard Markeloff for their valuable comments. The work of AG-M is supported by the SynBio3D project of the UK Engineering and Physical Sciences Research Council (EP/R019002/1) and the BioRoboost Contract of the European Union (820699). The work of CM is supported by the National Science Foundation under Grant

Table 2: Files that are referred to in the text and their locations in the SBOL-OWL GitHub repository.

	Description	File	Folder
1	RDFS reasoning	`JenaRDFSInferencingExample.java`	`sbol-sem/src/../examples`
2	PoPS receiver	`popsreceiver.rdf`	`examples`
3	PoPS receiver & SBOL-OWL	`popsreceiver_sbolowl.rdf`	`examples`
4	circuit_0x78 regulatory circuit	`circuit_0x78_environment_md.rdf`	`examples`
5	circuit_0x78 regulatory circuit with queries & SBOL-OWL	`circuit_0x78_environment_md_sbolowl_withqueries.rdf`	`examples`
6	sbol-10503 validation rule	`sbol-10503.xml`	`examples/validationrules_sbolowl`
7	sbol-10511 validation rule	`sbol-10511.xml`	`examples/validationrules_sbolowl`
8	sbol-10516 validation rule	`sbol-10516.xml`	`examples/validationrules_sbolowl`
9	sbol-10807 validation rule	`sbol-10807.xml`	`examples/validationrules_sbolowl`
10	sbol-11406 validation rule	`sbol-11406.xml`	`examples/validationrules_sbolowl`
11	sbol-11608 validation rule	`sbol-11608.xml`	`examples/validationrules_sbolowl`
12	2018 iGEM Distribution Plate 1 collection		`examples/igem/designs_chassis_sbolowl`
13	PoPS receiver with queries & SBOL-OWL	`popsreceiver_sbolowl_withqueries.rdf`	`examples`
14	Validating the 2018 iGEM Distribution Plate 1 collection	`SynBioHub_iGEM2018DistributionExample.java`	`sbol-sem/src/../examples/`
15	Executing semantic queries programmatically using semanticSBOL	`SemanticQueryingExample.java`	`sbol-sem/src/../examples/`

No. DBI-1356041, CCF-1748200, and DARPA FA8750-17-C-0229. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Supporting Information Available

Comparison of SBOL-OWL enabled semantic queries vs graph-based queries. This material is available free of charge via the Internet at `http://pubs.acs.org/`.

References

(1) Amos, M.; Goni-Moreno, A. Computational Matter; Springer, 2018; pp 93–110.

(2) Sprinzak, D.; Elowitz, M. B. Reconstruction of genetic circuits. Nature **2005**, 438, 443–448.

(3) Peccoud, J.; Anderson, J. C.; Chandran, D.; Densmore, D.; Galdzicki, M.; Lux, M. W.; Rodriguez, C. A.; Stan, G.-B.; Sauro, H. M. Essential information for synthetic DNA sequences. Nat. Biotechnol. **2011**, 29, 22.

(4) Myers, C. J.; Beal, J.; Gorochowski, T. E.; Kuwahara, H.; Madsen, C.; McLaughlin, J. A.; Mısırlı, G.; Nguyen, T.; Oberortner, E.; Samineni, M.; Wipat, A.; Zhang, M.; Zundel, Z. A standard-enabled workflow for synthetic biology. Biochem. Soc. Trans. **2017**, 45, 793–803.

(5) Goni-Moreno, A.; Carcajona, M.; Kim, J.; Martínez-García, E.; Amos, M.; de Lorenzo, V. An implementation-focused bio/algorithmic workflow for synthetic biology. ACS Synth. Biol. **2016**, 5, 1127–1135.

(6) Benson, D. A.; Cavanaugh, M.; Clark, K.; Karsch-Mizrachi, I.; Ostell, J.; Pruitt, K. D.; Sayers, E. W. GenBank. Nucleic Acids Res. **2018**, 46, D41–D47.

(7) Galdzicki, M.; Clancy, K. P.; Oberortner, E.; Pocock, M.; Quinn, J. Y.; Rodriguez, C. A.; Roehner, N.; Wilson, M. L.; Adam, L.; Anderson, J. C.; Bartley, B. A.; Beal, J.; Chandran, D.; Chen, J.; Densmore, D.; Endy, D.; Grunberg, R.; Hallinan, J.; Hillson, N. J.; Johnson, J. D.; Kuchinsky, A.; Lux, M.; Misirli, G.; Peccoud, J.; Plahar, H. A.; Sirin, E.; Stan, G.-B.; Villalobos, A.; Wipat, A.; Gennari, J. H.; Myers, C. J.; Sauro, H. M. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. Nat. Biotechnol. **2014**, 32, 545–550.

(8) Bartley, B.; Beal, J.; Clancy, K.; Misirli, G.; Roehner, N.; Oberortner, E.; Pocock, M.; Bissell, M.; Madsen, C.; Nguyen, T.; Zhang, Z.; Gennari, J. H.; Myers, C.; Wipat, A.; Sauro, H. Synthetic Biology Open Language (SBOL) Version 2.0.0. J. Integr. Bioinform. **2015**, 12, 272.

(9) Roehner, N.; Oberortner, E.; Pocock, M.; Beal, J.; Clancy, K.; Madsen, C.; Misirli, G.; Wipat, A.; Sauro, H.; Myers, C. J. Proposed Data Model for the Next Version of the Synthetic Biology Open Language. ACS Synth. Biol. **2015**, 4, 57–71.

(10) others,, et al. Targeted development of registries of biological parts. PLoS One **2008**, 3, e2671.

(11) Bilitchenko, L.; Liu, A.; Cheung, S.; Weeding, E.; Xia, B.; Leguia, M.; Anderson, J. C.; Densmore, D. Eugene A Domain Specific Language for Specifying and Constraining Synthetic Biological Parts, Devices, and Systems. PLoS One **2011**, 6, 1–12.

(12) Allemang, D.; Hendler, J. Semantic web for the working ontologist: effective modeling in RDFS and OWL; Elsevier, 2011.

(13) Galdzicki, M. The Synthetic Biology Open Language a data exchange standard for biological engineering. Ph.D. thesis, 2013.

(14) Zhang, Z.; Nguyen, T.; Roehner, N.; Misirli, G.; Pocock, M.; Oberortner, E.; Samineni, M.; Zundel, Z.; Beal, J.; Clancy, K.; Wipat, A.; Myers, C. J. libSBOLj 2.0: A Java Library to Support SBOL 2.0. IEEE Life Sci. Lett. **2016**, 1, 34–37.

(15) Bartley, B. A.; Choi, K.; Samineni, M.; Zundel, Z.; Nguyen, T.; Myers, C. J.; Sauro, H. M. pySBOL: A Python Package for Genetic Design Automation and Standardization. ACS Synth. Biol. **2018**,

(16) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Wilkinson, N.; Atallah, C.; Wipat, A. sboljs: Bringing the Synthetic Biology Open Language to the Web Browser. ACS Synth. Biol. **2019**, 8, 191–193.

(17) Shadbolt, N.; Berners-Lee, T.; Hall, W. The Semantic Web Revisited. IEEE Intell. Syst. **2006**, 21, 96–101.

(18) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Mısırlı, G.; Zhang, M.; Ofiteru, I. D.; Goni-Moreno, A.; Wipat, A. SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology. ACS Synth. Biol. **2018**, 7, 682–688.

(19) Madsen, C.; McLaughlin, J. A.; Misirli, G.; Pocock, M.; Flanagan, K.; Hallinan, J.; Wipat, A. The SBOL Stack: A Platform for Storing, Publishing, and Sharing Synthetic Biology Designs. ACS Synth. Biol. **2016**, 5, 487–497, PMID: 27268205.

(20) Antezana, E.; Kuiper, M.; Mironov, V. Biological knowledge management: the emerging role of the Semantic Web technologies. Briefings Bioinf. **2009**, 10, 392–407.

(21) Gruber, T. R. Toward principles for the design of ontologies used for knowledge sharing? Int. J. Man-Mach. Stud. **1995**, 43, 907–928.

(22) Stevens, R.; Aranguren, M. E.; Wolstencroft, K.; Sattler, U.; Drummond, N.; Horridge, M.; Rector, A. Using OWL to model biological knowledge. Int. J. Man-Mach. Stud. **2007**, 65, 583–594.

(23) Consortium, G. O. Gene ontology consortium: going forward. Nucleic Acids Res. **2014**, 43, D1049–D1056.

(24) Antezana, E.; Kuiper, M.; Mironov, V. Biological knowledge management: the emerging role of the Semantic Web technologies. Briefings Bioinf. **2009**, 10, 392–407.

(25) Blondé, W.; Mironov, V.; Venkatesan, A.; Antezana, E.; De Baets, B.; Kuiper, M. Reasoning with bio-ontologies: using relational closure rules to enable practical querying. Bioinformatics **2011**, 27, 1562–1568.

(26) Madsen, C.; Moreno, A.; Nguyen, T.; Palchick, Z.; Roehner, N.; Wipat, A.; Beal, J.; Bissell, M.; Gennari, J. H.; Sauro, H.; Wipat, A. Synthetic Biology Open Language (SBOL) Version 2.2.0. J. Integr. Bioinform. **2018**, 15.

(27) Demir, E.; Cary, M. P.; Paley, S.; Fukuda, K.; Lemer, C.; Vastrik, I.; Wu, G.; D'Eustachio, P.; Schaefer, C.; Luciano, J.; Schacherer, F.; Martinez-Flores, I.; Hu, Z.; Jimenez-Jacinto, V.; Joshi-Tope, G.; Kandasamy, K.; Lopez-Fuentes, A. C.; Mi, H.; Pichler, E.; Rodchenkov, I.; Splendiani, A.; Tkachev, S.; Zucker, J.; Gopinath, G.; Rajasimha, H.; Ramakrishnan, R.; Shah, I.; Syed, M.; Anwar, N.; Babur, Ö.; Blinov, M.; Brauner, E.; Corwin, D.; Donaldson, S.; Gibbons, F.; Goldberg, R.; Hornbeck, P.; Luna, A.; Murray-Rust, P.; Neumann, E.; Ruebenacker, O.; Samwald, M.; van Iersel, M.; Wimalaratne, S.; Allen, K.; Braun, B.; Whirl-Carrillo, M.; Cheung, K.-H.; Dahlquist, K.; Finney, A.; Gillespie, M.; Glass, E.; Gong, L.; Haw, R.; Honig, M.; Hubaut, O.; Kane, D.; Krupa, S.; Kutmon, M.; Leonard, J.; Marks, D.; Merberg, D.; Petri, V.; Pico, A.; Ravenscroft, D.; Ren, L.; Shah, N.; Sunshine, M.; Tang, R.; Whaley, R.; Letovksy, S.; Buetow, K. H.; Rzhetsky, A.; Schachter, V.; Sobral, B. S.; Dogrusoz, U.; McWeeney, S.; Aladjem, M.; Birney, E.; Collado-Vides, J.; Goto, S.; Hucka, M.; Le Novère, N.; Maltsev, N.; Pandey, A.; Thomas, P.; Wingender, E.; Karp, P. D.; Sander, C.; Bader, G. D. The BioPAX community standard for pathway data sharing. Nat. Biotechnol. **2010**, 28, 935–942.

(28) Glimm, B.; Horrocks, I.; Motik, B.; Stoilos, G.; Wang, Z. HermiT: an OWL 2 reasoner. J. Autom. Reason. **2014**, 53, 245–269.

(29) Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; Katz, Y. Pellet: A practical owl-dl reasoner. Web Semant. **2007**, 5, 51–53.

(30) Tsarkov, D.; Horrocks, I. FaCT++ description logic reasoner: System description. International Joint Conference on Automated Reasoning. 2006; pp 292–297.

(31) Horrocks, I.; Patel-Schneider, P. F.; Van Harmelen, F. From SHIQ and RDF to OWL: The making of a web ontology language. Web Semant. **2003**, 1, 7–26.

(32) Fowler, M. UML distilled: a brief guide to the standard object modeling language; Addison-Wesley Professional, 2004.

(33) Eilbeck, K.; Lewis, S. E.; Mungall, C. J.; Yandell, M.; Stein, L.; Durbin, R.; Ashburner, M. The Sequence Ontology: a tool for the unification of genome annotations. Genome Biol. **2005**, 6, R44.

(34) Ison, J.; Kalaš, M.; Jonassen, I.; Bolser, D.; Uludag, M.; McWilliam, H.; Malone, J.; Lopez, R.; Pettifer, S.; Rice, P. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. Bioinformatics **2013**, 29, 1325–1332.

(35) Juty, N.; le Novère, N. Systems biology ontology. Encyclopedia of Systems Biology **2013**, 2063–2063.

(36) Horridge, M.; Patel-Schneider, P. F. OWL 2 Web Ontology Language Manchester syntax. 2009; `https://www.w3.org/TR/owl2-manchester-syntax/`.

(37) Canton, B.; Labno, A.; Endy, D. Refinement and standardization of synthetic biological parts and devices. Nat. Biotechnol. **2008**, 26, 787.

(38) Casteleiro, M. A.; Klein, J.; Stevens, R. The proteasix ontology. J. Biomed. Semantics **2016**, 7, 33.

(39) Misirli, G.; Nguyen, T.; McLaughlin, J. A.; Vaidyanathan, P.; Jones, T. S.; Densmore, D.; Myers, C.; Wipat, A. A Computational Workflow for the Automated Generation of Models of Genetic Designs. ACS Synth. Biol. **2018**,

(40) Quinn, J. Y.; Cox, R. S., III; Adler, A.; Beal, J.; Bhatia, S.; Cai, Y.; Chen, J.; Clancy, K.; Galdzicki, M.; Hillson, N. J.; Le Novre, N.; Maheshwari, A. J.; McLaughlin, J. A.; Myers, C. J.; P, U.; Pocock, M.; Rodriguez, C.; Soldatova, L.; Stan, G.-B. V.; Swainston, N.; Wipat, A.; Sauro, H. M. SBOL Visual: A Graphical Language for Genetic Designs. PLoS Biol. **2015**, 13, e1002310.

(41) Stein, L. D. Integrating biological databases. Nat. Rev. Genet. **2003**, 4, 337.

(42) Glimm, B.; Krötzsch, M. SPARQL beyond subgraph matching. International Semantic Web Conference. 2010; pp 241–256.

(43) Sirin, E.; Parsia, B. SPARQL-DL: SPARQL Query for OWL-DL. OWLED. 2007.

(44) Misirli, G.; Hallinan, J.; Pocock, M.; Lord, P.; McLaughlin, J. A.; Sauro, H.; Wipat, A. Data Integration and Mining for Synthetic Biology Design. ACS Synth. Biol. **2016**, 5, 1086–1097.

(45) Misirli, G.; Hallinan, J.; Wipat, A. Composable Modular Models for Synthetic Biology. ACM J. Emerg. Technol. Comput. Syst. **2014**, 11, 22:1–22:19.

(46) Misirli, G.; Cavaliere, M.; Waites, W.; Pocock, M.; Madsen, C.; Gilfellon, O.; Honorato-Zimmer, R.; Zuliani, P.; Danos, V.; Wipat, A. Annotation of rule-based models with formal semantics to enable creation, analysis, reuse and visualization. Bioinformatics **2016**, 32, 908–917.

(47) Neal, M. L.; Knig, M.; Nickerson, D.; Msrl, G.; Kalbasi, R.; Drger, A.; Atalag, K.; Chelliah, V.; Cooling, M. T.; Cook, D. L.; Crook, S.; de Alba, M.; Friedman, S. H.;

Garny, A.; Gennari, J. H.; Gleeson, P.; Golebiewski, M.; Hucka, M.; Juty, N.; Myers, C.; Olivier, B. G.; Sauro, H. M.; Scharm, M.; Snoep, J. L.; Tour, V.; Wipat, A.; Wolkenhauer, O.; Waltemath, D. Harmonizing semantic annotations for computational models in biology. Briefings Bioinf. **2018**, bby087.

(48) Neal, M. L.; Thompson, C. T.; Kim, K. G.; James, R. C.; Cook, D. L.; Carlson, B. E.; Gennari, J. H. SemGen: a tool for semantics-based annotation and composition of biosimulation models. Bioinformatics **2018**, bty829.

(49) Roehner, N.; Myers, C. J. A methodology to annotate systems biology markup language models with the synthetic biology open language. ACS Synth. Biol. **2013**, 3, 57–66.

(50) Lord, P. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. Experiences and Directions Workshop (OWLED). Montpellier, France, 2013.

(51) Emerick, C.; Carper, B.; Grand, C. Clojure Programming: Practical Lisp for the Java World; "O'Reilly Media, Inc.", 2012.

(52) Peroni, S.; Shotton, D.; Vitali, F. The Live OWL Documentation Environment: a tool for the automatic generation of ontology documentation. International Conference on Knowledge Engineering and Knowledge Management. 2012; pp 398–412.

(53) Zhang, Z.; Nguyen, T.; Roehner, N.; Misirli, G.; Pocock, M.; Oberortner, E.; Samineni, M.; Zundel, Z.; Beal, J.; Clancy, K.; Wipat, A.; Myers, C. J. libSBOLj 2.0: A Java Library to Support SBOL 2.0. IEEE Life Sci. Lett. **2015**, 1, 34–37.

(54) Horridge, M.; Bechhofer, S. The owl api: A java api for owl ontologies. Semantic Web **2011**, 2, 11–21.

Graphical TOC Entry